UNITED STATES PATENT APPLICATION

for

A STREAMED DATABASE ARCHIVAL PROCESS WITH BACKGROUND SYNCHRONIZATION

Inventor:

William Williams

prepared by:

WAGNER, MURABITO & HAO
Two North Market Street
Third Floor
San Jose, CA 95113
(408) 938-9060

A STREAMED DATABASE ARCHIVAL PROCESS WITH BACKGROUND SYNCHRONIZATION

FIELD OF THE INVENTION

The present invention relates to an improved archival process for facilitating database backup. More particularly, the present invention pertains to a streamed database archival process with background synchronization.

BACKGROUND OF THE INVENTION

Databases are used for accepting, storing, and providing, on demand, data for multiple users. A database serves as a central repository of data which can be accessed by any number of different computers and users at any time. Databases find use in many different fields, such as for storing medical information, financial transactions, web sites, etc. Indeed, databases are becoming crucial to the operation of many companies. For example, a production database is often used by a company to track orders placed by customers, control inventory, monitor the workflow during assembly, generate invoices, etc.

Unfortunately, databases are susceptible to becoming disabled due to hardware breakdowns, software glitches, network hangups, accidents, disasters, human error, etc. In many instances, it would be catastrophic if

data could not be accessed, or worse, if data were to be permanently lost due to a database failure. Because there can never be a guarantee that a database is always up and running, companies have implemented backup databases. And depending on the degree of importance of the data or function of a particular database, several different copies of the same data are often archived in multiple databases at different geographical locations on different storage facilities. In case of a failure in the primary operational database, the company can rely on the backup database. The company can switch from the primary operational database to the backup database in a matter of minutes.

Since data in the primary database is constantly being added, deleted, or otherwise modified, the backup database must likewise be updated so that it contains a mirror copy of the data residing on the primary database. Rather than copying over the entire database each and every time the operational database is updated, the backup database is periodically updated by copying over only the new changes since the previous update. For instance, the operational database can create an archive log which is used to store a pre-determined number of new transactions. When the archive log is full, that archive log is sent to the backup database. The backup database stores the archive log. In this manner, the backup database is kept current.

Although this archive process is simple and straightforward, it suffers from several drawbacks. One very important issue is that if a database administrator inadvertently removes a log on the receiving host, it will need to be manually recopied because prior art processes would only copy a log

once and then move on. There is no mechanism for recopying. A related issue is that there is an inherent amount of danger in corrupting or deleting a log file as it is being manually recopied. In that event, the backup database on the receiving side is nullified and needs to be fully recopied. This is highly undesirable as it causes downtime for clients and a copious amount of human labor must be exerted just to return service to the application.

Another issue with the prior art archiving processes is that they could only sequentially transfer one log at a time. If there were a backlog or a delay (e.g., loss of network connectivity), logs would pile up on the sending host, thereby resulting in the data on the backup database becoming more and more stale. There is no real way to recoup from a backlog other than to manually copy many logs in batch mode and risking data loss or corruption through human error.

Therefore, there exists a need in the prior art for an improved archiving process. The present invention provides a unique, novel solution to these and other archival problems.

SUMMARY OF THE INVENTION

The present invention pertains to streamed database archival process with background synchronization. An operational host database saves all transactions in an archive log for backup purposes. When a number of archive logs have been saved, they are transferred to a backup database which resides on a different device at a geographically different site. These archive logs are streamed asynchronously such that multiple logs can be transferred simultaneously from the host operational database to the backup archive database. By streaming the logs, the backup database is kept constantly updated and backlogs can quickly be resolved. Furthermore, an automatic recovery process is run in the background. This process automatically detects files on the backup database which may have been accidentally deleted or corrupted by comparing file systems of the host database to that of the backup database. The process recopies those missing or corrupted files from the host database to the backup database without any human intervention.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

Figure 1 shows an archival system upon which the present invention may be practiced.

Figure 2 shows a flowchart describing the steps for performing an asynchronous streamed archive log transfer according to one embodiment of the present invention.

Figure 3 shows a flowchart describing the steps for performing automatic recovery according to one embodiment of the present invention.

Figure 4 shows an exemplary computer system upon which embodiments of the present invention may be practiced.

DETAILED DESCRIPTION

An improved archival process for facilitating database backup is described. Specifically, a streamed database archival process with background synchronization is disclosed. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be obvious, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid obscuring the present invention.

Referring to Figure 1, an archival system upon which the present invention may be practiced is shown. A local area network (LAN) 101 is used to route data between a number of computers 102-104 coupled to the LAN. A primary operational database 105 residing on a mass storage device (e.g., disk array) is also coupled to LAN 101. Thereby, any number of users can access operational database 105 via LAN 101. An exemplary database is Oracle 7 relational data base. In addition, the users can gain access to the Internet 106 via LAN 101. Another local area network/metropolitan area network (LAN/MAN) 107 coupled to the Internet 106 provides networking between a number of users 108-111. A backup database 112 is coupled to LAN/MAN 112. The backup database 112 is situated at a different geographical location than the operational database 105. The backup database 112 contains an archived copy of the complete set of data residing on operational database

105. New transactions stored on operational database 105 are stored as files in an archive log. The archive log is then transferred from the host operational database 105 to the receiving backup database 112. The present invention pertains to the process by which the log transfers are handled and is described in detail below. The transfers can be made through a company's intranet, a virtual private network, or on a dedicated link. Optionally, one or more additional backup databases, such as backup database 113, can be set up and maintained. If operational database 105 were to fail, the backup database (e.g., either backup database 112 or 113) can be transitioned to function as the new operational database.

In the currently preferred embodiment, archive logs are transferred from the host database to the receiving backup database in the form of multiple asynchronous streams. In other words, rather than queuing logs to be sent sequentially whereby one archive log is sent and then waiting for that transfer to complete before the next archive log can be sent, the present invention prepares the archive logs such that multiple archive logs can be transferred simultaneously. This is accomplished by setting up multiple streams, whereby each stream transfers one archive log. The archive logs can be sent asynchronously over multiple simultaneous streams. The advantage of utilizing multiple asynchronous streams is that if the backup process falls behind, it can quickly and expeditiously catch back up. In other words, suppose that a router goes down and archive logs build up on the host database because they cannot be sent to the backup database due to the failed router. Once the network is fixed, the host database can send multiple

streams of archive logs. This means that the backup database can be updated much faster than before. The significance is that the archived data does not become too stale. If the host database were to fail, the backup database would more closely mirror the host database.

Figure 2 is a flowchart describing the steps for performing an asynchronous streamed archive log transfer according to one embodiment of the present invention. The first step 201 entails opening a session. A config file is read to determine user-specified options, step 202. These user-specified options could include the number of separate streams to transfer at a time, who to contact in case of an error, the host, the receiving destination, as well as address information). Based on the data read from the config file, the process reads the specified directory and obtains the appropriate archive log files, step 203. An array of all the specified archive log files which need to be transferred is then built, step 204. The specified number of streams are set up to transfer the archive log files, step 205. The archived log files are written out in multiple, simultaneous, asynchronous streams, step 206. This process repeats until all archived logs in the array have been transferred, step 207. It should be noted that towards the end of the process, streams may be sent sequentially once there are not enough archive logs left in the array to support multiple simultaneous streams. Moreover, this process is optimally run automatically in cron. Alternatively, it can be run by hand.

In one specific implementation, the process is written in PERL. The PERL program wakes up and reads a config file. It then shells over to the

identified remote host and reads the appropriate file (e.g., thread1.dat) to obtain a starting number. An array of archive logs is then built to ship based on the delta between the starting number (e.g., thread1.dat) and the number of the last log created minus one (n-1). If compress is specified, the archive logs are zipped. The specified number of streams are then set up to rysnc transfer the archived logs.

In one embodiment of the present invention, automatic recovery of corrupted and/or missing archive logs is provided. Once an archive log has been copied over to the backup database, it may become corrupted. Occasionally, one of the archive logs stored on the backup database inadvertently gets deleted. In the past, the system administrator would have to research the problem, figure out which archive log was corrupted or missing and then manually recopy that archive log from the host database. With the present invention, this is all done automatically in the background. The present invention regularly checks the archived logs stored in the backup database against the archived logs maintained in the host database. If there is a discrepancy, the archive log at issue is recopied from the host database to the backup database. Thereby, this eliminates the need for any human intervention.

Figure 3 is a flowchart describing the steps for performing automatic recovery according to one embodiment of the present invention. Upon start up, the process performs a differential check, step 301. It checks the local directory corresponding to the backup database against the host directory

corresponding to the operational database. Based on the comparison between these two file systems, the process can determine whether an archive log is missing, step 302. Even though that particular archive log had already been copied over once before, it might have been deleted. The program automatically issues a command to instruct the host to transfer that particular archive log, step 303. That particular archive log is immediately transferred by the host even though the host had already previously transferred that archive log. In addition, based on the comparison between the host and local file systems, the process can determine whether there is a discrepancy in the size of any of the archive logs, step 304. This may be accomplished by performing a checksum. If the log size is different, this indicates that the log has become corrupted. The process issues a command to instruct the host to immediately transfer that particular log, step 305. The log is immediately transferred and copied onto the backup database. The process continues to periodically perform differential checks in the background, step 301. Thereby, the present invention can automatically catch and fix problems which may arise without any human intervention.

In the currently preferred embodiment, the present invention is implemented as a process and a PERL program known as Sync_arch. It utilizes standard streaming, rsh or ssh (for secure data transfer), and rsync, which are publicly available tools. Sync_arch is highly configurable and easy to install. It allows for configuration of multiple copy streams, which prevents backlogs, and enables recovery from backlogs. The use of rsync and its rolling checksum mechanism prevents log file corruption and verifies the

integrity of the data. The algorithm behind Sync_arch allows for automatic recopying of logs that have been inadvertently removed, preventing operator intervention. Thereby, Sync_arch provides secure and fast data transfer that recovers quickly from errors and helps keep production and disaster databases in sync.

Referring now to Figure 4, an exemplary computer system 490 upon which embodiments of the present invention may be practiced is shown. In general, computer system 490 comprises bus 400 for communicating information, processor 401 coupled with bus 400 for processing information and instructions, random access (volatile) memory 402 coupled with bus 400 for storing information and instructions for processor 401, read-only (non-volatile) memory 403 coupled with bus 400 for storing static information and instructions for processor 401. A data storage device 404 such as a magnetic or optical disk and disk drive is coupled with bus 400 for storing information and instructions, such as the Sync_arch program. An optional user output device such as display device 405 is coupled to bus 400 for displaying information to the computer user. Computer system 490 further comprises an optional user input device such as alphanumeric input device 406 including alphanumeric and function keys coupled to bus 400 for communicating information and command selections to processor 401, and an optional user input device such as cursor control device 407 coupled to bus 400 for communicating user input information and command selections to processor 401. Furthermore, a network interface card (NIC) 408 is used to

couple computer system 490 onto, for example, a client-server computer system network. In such a network, computer system 490 can exemplify a client computer system and/or a server computer system.

Display device 405 utilized with computer system 490 may be a liquid crystal device, cathode ray tube, or other display device suitable for creating graphic images and alphanumeric characters recognizable to the user. Cursor control device 407 allows the computer user to dynamically signal the two-dimensional movement of a visible symbol (pointer) on a display screen of display device 405. Many implementations of the cursor control device are known in the art including a trackball, mouse, joystick or special keys on alphanumeric input device 406 capable of signaling movement of a given direction or manner of displacement. It is to be appreciated that the cursor control 407 also may be directed and/or activated via input from the keyboard using special keys and key sequence commands. Alternatively, the cursor may be directed and/or activated via input from a number of specially adapted cursor directing devices.

Therefore, a streamed database archival process with background synchronization is disclosed. The foregoing descriptions of specific embodiments of the present invention have been presented for purposes of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed, and obviously many modifications and variations are possible in light of the above teaching. The

embodiments were chosen and described in order to best explain the principles of the invention and its practical application, to thereby enable others skilled in the art to best utilize the invention and various embodiments with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the Claims appended hereto and their equivalents.